1. Make a class called Thing with no contents and print it. Then, create an object called example from this class and also print it. Are the printed values the same or different?

Solution:

```python

class Thing:

pass

print(Thing())

example = Thing()

print(example)

```

Output:

```

<\_\_main\_\_.Thing object at 0x7f9767f3a0a0>

<\_\_main\_\_.Thing object at 0x7f9767f3a1c0>

```

The printed values are different.

2. Create a new class called Thing2 and add the value 'abc' to the letters class attribute. Letters should be printed.

Solution:

```python

class Thing2:

letters = 'abc'

print(Thing2.letters)

```

Output:

```

abc

```

3. Make yet another class called, of course, Thing3. This time, assign the value 'xyz' to an instance (object) attribute called letters. Print letters. Do you need to make an object from the class to do this?

Solution:

```python

class Thing3:

def \_\_init\_\_(self):

self.letters = 'xyz'

print(Thing3().letters)

```

Output:

```

xyz

```

4. Create an Element class with the instance attributes name, symbol, and number. Create a class object with the values 'Hydrogen,' 'H,' and 1.

Solution:

```python

class Element:

def \_\_init\_\_(self, name, symbol, number):

self.name = name

self.symbol = symbol

self.number = number

el = Element('Hydrogen', 'H', 1)

```

5. Make a dictionary with these keys and values: 'name': 'Hydrogen', 'symbol': 'H', 'number': 1. Then, create an object called hydrogen from class Element using this dictionary.

Solution:

```python

el\_dict = {'name': 'Hydrogen', 'symbol': 'H', 'number': 1}

hydrogen = Element(\*\*el\_dict)

```

6. For the Element class, define a method called dump() that prints the values of the object’s attributes (name, symbol, and number). Create the hydrogen object from this new definition and use dump() to print its attributes.

Solution:

```python

class Element:

def \_\_init\_\_(self, name, symbol, number):

self.name = name

self.symbol = symbol

self.number = number

def dump(self):

print(f'{self.name} ({self.symbol}), number {self.number}')

hydrogen = Element(\*\*el\_dict)

hydrogen.dump()

```

Output:

```

Hydrogen (H), number 1

```

7. Call print(hydrogen). In the definition of Element, change the name of method dump to \_\_str\_\_, create a new hydrogen object, and call print(hydrogen) again.

Solution:

```python

class Element:

def \_\_init\_\_(self, name, symbol, number):

self.name = name

self.symbol = symbol

self.number = number

def \_\_str\_\_(self):

return f'{self.name} ({self.symbol}), number {self.number}'

hydrogen = Element(\*\*el\_dict)

print(hydrogen)

```

Output:

```

Hydrogen (H), number 1

```

8. Modify Element to make the attributes name, symbol, and number private. Define a getter property for each to return its value.

Solution:

```python

class Element:

def \_\_init\_\_(self, name, symbol, number):

self.\_\_name = name

self.\_\_symbol = symbol

self.\_\_number = number

def get\_name(self):

return self.\_\_name

def get\_symbol(self):

return self.\_\_symbol

def get\_number(self):

return self.\_\_number

hydrogen = Element(\*\*el\_dict)

print(hydrogen.get\_name())

print(hydrogen.get\_symbol())

print(hydrogen.get\_number())

```

Output:

```

Hydrogen

H

1

```

9. Define three classes: Bear, Rabbit, and Octothorpe. For each, define only one method: eats(). This should return 'berries' (Bear), 'clover' (Rabbit), or 'campers' (Octothorpe). Create one object from each and print what it eats.

Solution:

```python

class Bear:

def eats(self):

return 'berries'

class Rabbit:

def eats(self):

return 'clover'

class Octothorpe:

def eats(self):

return 'campers'

bear = Bear()

rabbit = Rabbit()

octothorpe = Octothorpe()

print(bear.eats())

print(rabbit.eats())

print(octothorpe.eats())

```

Output:

```

berries

clover

campers

```

10. Define these classes: Laser, Claw, and SmartPhone. Each has only one method: does(). This returns 'disintegrate' (Laser), 'crush' (Claw), or 'ring' (SmartPhone). Then, define the class Robot that has one instance (object) of each of these. Define a does() method for the Robot that prints what its component objects do.

Solution:

```python

class Laser:

def does(self):

return 'disintegrate'

class Claw:

def does(self):

return 'crush'

class SmartPhone:

def does(self):

return 'ring'

class Robot:

def \_\_init\_\_(self):

self.laser = Laser()

self.claw = Claw()

self.smartphone = SmartPhone()

def does(self):

print(f'Laser: